**Parte 3:**

**1. ¿Por qué eligieron ese ORM y qué beneficios o dificultades encontraron?**

Elegimos **Prisma** porque ofrece un esquema declarativo (schema.prisma), migraciones automáticas y un cliente tipado que genera autocompletado. Los beneficios fueron: velocidad de desarrollo, seguridad de tipos y migraciones versionadas. La principal dificultad fue adaptar validaciones avanzadas (CHECK) mediante migraciones manuales, pues Prisma todavía no soporta @@check directo en el esquema.

**2. ¿Cómo implementaron la lógica master-detail dentro del mismo formulario?**  
Usamos un único <form> con campos de **Student** y un <select multiple name="courseIds">. Al recibir el POST, Prisma crea el estudiante y, en la misma operación, inserta las filas en la tabla intermedia con:

js

CopiarEditar

prisma.student.create({

data: {

/\* campos de estudiante \*/,

enrollments: {

create: courseIds.map(id => ({ courseId: +id }))

}

}

});

**3. ¿Qué validaciones implementaron en la base de datos y cuáles en el código?**

* **Base de datos**:
  + NOT NULL en campos obligatorios.
  + UNIQUE en email y code.
  + CHECK (longitud mínima, regex de email, fecha ≤ now).
  + ON DELETE CASCADE en las FKs.
* **Código (Express/EJS + HTML5)**:
  + HTML5 (required, minlength, type="email").
  + Captura de excepciones de Prisma en un middleware global para retornar errores.

**4. ¿Qué beneficios encontraron al usar tipos de datos personalizados?**  
Los **ENUMs** (gender, course\_level) garantizan que solo se inserten valores válidos, simplifican validaciones y Prisma los refleja como tipos TS, evitando errores de asignación en tiempo de compilación.

**5. ¿Qué ventajas ofrece usar una VIEW como base del índice en vez de una consulta directa?**

* Encapsula la lógica JOIN entre tres tablas.
* Simplifica el código de la ruta (SELECT \* FROM StudentCourseView).
* Centraliza mantenimiento y promueve reuso (cualquier cambio en la presentación solo modifica la VIEW).

**6. ¿Qué escenarios podrían romper la lógica actual si no existieran las restricciones?**

* Sin UNIQUE(email), podrían crearse estudiantes duplicados.
* Sin ON DELETE CASCADE, al eliminar un estudiante quedarían inscripciones huérfanas.
* Sin la **PK compuesta** de Enrollment, se podrían duplicar pares (studentId, courseId).

**7. ¿Qué aprendieron sobre la separación entre lógica de aplicación y lógica de persistencia?**  
Mantener el **modelo de datos** en Prisma y las migraciones junto al esquema de BD nos permitió separar claramente el **acceso** (Prisma Client) de la **lógica de negocio** (Express). Así, la app no necesita conocer detalles del DDL, solo invoca métodos de Prisma.

**8. ¿Cómo escalaría este diseño en una base de datos de gran tamaño?**

* Añadiendo **paginación** a findMany() y límites en las consultas.
* Creando **índices** adicionales si surgen nuevos filtros.
* Optimizando la VIEW o reemplazándola por **materialized views** para cargas muy pesadas.

**9. ¿Consideran que este diseño es adecuado para una arquitectura con microservicios?**  
Sí:

* El servicio de **usuarios/estudiantes** podría exponer solo el CRUD de Student/Enrollment.
* Otro microservicio podría consumir la VIEW o endpoints especializados.
* Prisma facilita definir múltiples **datasources** si cada microservicio usa su propia BD.

**10. ¿Cómo reutilizarían la vista en otros contextos como reportes o APIs?**

* En un endpoint REST/GraphQL se puede exponer /api/enrollments que consulte directamente la VIEW con $queryRaw.
* Para reportes PDF o dashboards, bastaría apuntar a la VIEW sin reescribir joins.

**11. ¿Qué decisiones tomaron para estructurar su modelo de datos y por qué?**

* **Entidad Student** separada de **Course** con tabla intermedia para N:M.
* **Enums** para validar género y nivel.
* **PK compuesta** en Enrollment para evitar duplicados.  
  Esto refleja la realidad académica de inscripciones, mantiene integridad y es fácil de entender.

**12. ¿Cómo documentaron su modelo para facilitar su comprensión por otros desarrolladores?**

* Añadimos **comentarios** SQL (COMMENT ON TABLE … y columnas) en schema.sql.
* El schema.prisma sirve de “mínimo común denominador” para entender entidades y relaciones.
* El README del repositorio describe el dominio, pasos de migración y uso de Prisma.

**13. ¿Cómo evitaron la duplicación de registros o errores de asignación en la tabla intermedia?**

* La **PK compuesta** (studentId, courseId) impide pares duplicados.
* Al editar usamos primero deleteMany y luego create, asegurando que la BD refleje exactamente la selección actual.
* Las ENUMs y validaciones CHECK evitan valores fuera de rango.

**Capturas de evidencia Funcional:**

Interfaz de usuario gráfica, Tabla

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Tabla

El contenido generado por IA puede ser incorrecto.